Introduction to Graphics

Assignment 5: Bezier Curves

Student: Kasper Passov

1 Bezier curves

A Bezier curve is created by two matrices, a *Basic matrix* and a *Geometry matrix*. The *Geometry matrix* contains 4 vectors called *Control Points*, descriping the placement and shape of the matrix. The first and fourth vector descripes the start and endpoint of the curve, where the secound and third are points along the curve, not necessarily on the curve. The *Basic matrix* is used to transform the curve into a Hermit curve.

2 Implementation

I implemteted all types of line calculations in the mouseBeizer framework.

2.1 Sampling

The sampling was implemented in the given framework, so no work was done on it.

```
Point drawBezierSample(Point A, Point B, Point C, Point D, double t)
{
    Point P;
    P.x = pow((1 - t), 3) * A.x + 3 * t * pow((1 -t), 2) * B.x +
             3 * (1-t) * pow(t, 2)* C.x + pow (t, 3)* D.x;
    P.y = pow((1 - t), 3) * A.y + 3 * t * pow((1 -t), 2) * B.y +
             3 * (1-t) * pow(t, 2)* C.y + pow (t, 3)* D.y;
        P.z = 0;
    return P;
```

The sampling is a naive implementation of the equation for a parametric curve

$$f(t) = at^3 + bt^2 + ct + d$$

Where the linesegments are calculated by the for loop in the main function

```
for(double t = 0.0;t <= 1.0; t += 1.0/n) {
    Point P = drawBezierSample(abc[0], abc[1], abc[2], abc[3],  t);
    drawLine(POld, P);
    POld = P;
```

This loop handles the increments of t, so we have calculated points equaly constributed along the curve, with $1/n$ between each point.

2.2 Forward Differencing

Using forward differencing we can reduce the cost of evaluation to only 3 additions per curvepoint. By using δ we can remove the t variable, replacing it with the variable that defines the interval of the point calculations. This means

we can replace the variable t with the constant δ making the calculations a lot simpler.
The matrix is defined in the function

```
void forwardDiffInit(Point A0, Point B0, Point C0, Point D0, double del, Point deltas[]){
    Point fda = (A0 * -1.0) + (B0 *  3.0) + (C0 * -3.0) + D0;
    Point fdb = (A0 *  3.0) + (B0 * -6.0) + (C0 *  3.0);
    Point fdc = (A0 * -3.0) + (B0 *  3.0);
    Point fdd =  A0;

    deltas[0] = fdd;
    deltas[1] = fda *     pow(del, 3.0) + fdb * pow(del, 2.0) + fdc * del;
    deltas[2] = fda * 6.0 * pow(del, 3.0) + fdb * 2.0 * pow(del, 2.0);
    deltas[3] = fda * 6.0 * pow(del, 3.0);
```

With this we can calculates the points using the itteration defined in the following function

```
point drawbezierforwarddiff(point deltas[]) {
    point p;
    p = deltas[0];

    deltas[0] = deltas[0] + deltas[1];
    deltas[1] = deltas[1] + deltas[2];
    deltas[2] = deltas[2] + deltas[3];
        return p;
```

The itterations of t is much the same, but with the addition of the initiation of the delta vector.

2.3 Subdivision

In my implementation of subdivision i ran into one large problem i am yet to solve. After calculating a point on the middle of the curve recursivly, i need to sort these points, so the points are connected in the correct order. This is a problem as the algorithm does not calculate them in the correct order. This i tried to get around by creating a recursive function, that creates a tree structure forcing the leftmost calculated point on the curve to be calculated first.

```
void drawBezierSubdivision(Point G1, Point G2, Point G3, Point G4, double n) {
    ...
    drawBezierSubdivision(Left[0], Left[1], Left[2], Left[3], floor(n/2));
    P = Left[0];
    drawLine(POld, P);
    POld = P;
    drawBezierSubdivision(Right[0], Right[1], Right[2], Right[3], ceil(n/2)-1);
```

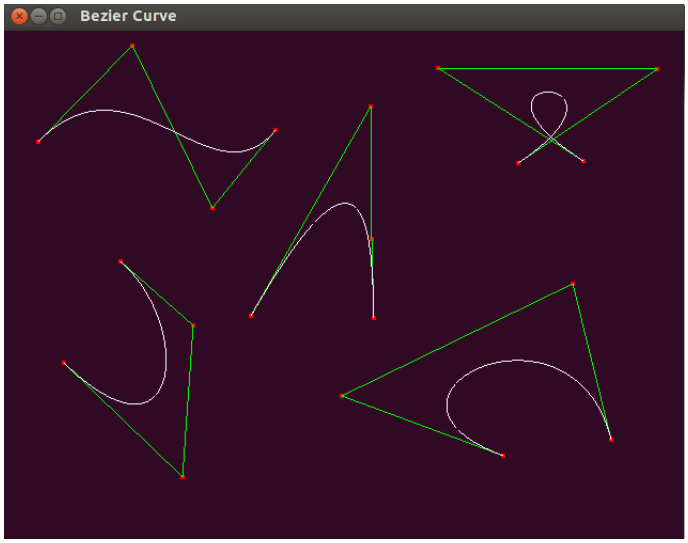
The values of the Left and Right vector is calculated above the recursive call. I ensure the right amount of points are calculated by checking ageinst a n value

that each recursive call get half of. If a call gets a value of 0, the point is not drawn.

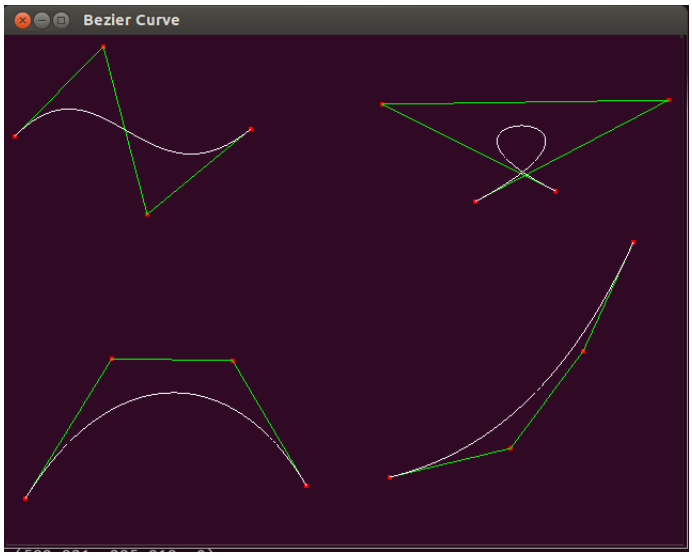
3 Testing

I created some curves with each of the 3 methods. Further testing can be done making and running the zipped file.

3.1 Sampling



3.2 Forwarding



3.3 Subdivision